VISIT…

Optimisation Methods in *SPICE*, a Comparison

Janez PUHAN* Tadej TUMA* Iztok FAJFAR*

Abstract

The field of optimisation is still not appropriately covered in modern computer programs for circuit analysis as *SPICE*. It was originally developed at the University of California at Berkeley. The last official release of *SPICE* 3f4 was published in 1992. It has numerous bugs, mainly memory leaks. In order to implement optimisation methods most bugs were tracked down and fixed.

Several well-known optimisation methods were implemented. The new command *optimize* was added to the interactive command interpreter called *Nutmeg* to manipulate with optimisation tasks. The *optimize* command represents a general optimisation tool, which can be used on any circuit with arbitrary cost function. The syntax of the command is introduced in the paper. Further five test circuits were optimised with different methods and a comparison between the implemented methods is presented. The essential properties of the optimisation methods in real optimisation problems are shown.

All simulation files as well as the new compilation of *SPICE* may be downloaded from *http://fides.fe.uni-lj.si/spice/*.

1 Introduction

Although Berkeley has stopped the development and maintenance of it's popular *SPICE* program in 1992 the freely available code has seen many different compilations ever since and is still the widest spread analog circuit simulator. Today there are commercial versions like *ISPICE* from Intusoft (*http://www.intusoft.com*) or *PSPICE* from OrCad (*http://www.orcad.com*) but there are also many freeware compilations for the Windows as well as the Linux platform. All this different products have one thing in common: their numerical core is more or less the untouched Berkeley pre-ANSI C code from 1992. The numerical methods implemented by Berkeley are very effective and still unsurpassed in their sophistication. There are great differences however in the graphical user interface, the libraries, the documentation and the technical support.

We are also developing and supporting our own *SPICE* compilation (*http://www.fe.uni-lj.si/spice*) with the main purpose of adding a general analog optimisation tool. We first had to build our own GUI then we ran into numerous memory leaks and other bugs. These bugs are intrinsic to any Berkeley *SPICE* derivative and are usually harmless, just wasting memory. In optimisation loops however thousands of different analyses must be run, so the wasted memory becomes a real problem. We didn't find a single compilation, which could run the following simple script without constantly loosing memory:

```
while 1
  rusage
end
```

The while loop should run forever reporting the system memory usage. There is no reason why the reported number constantly increases. Memory leaks are not bound to this particular script, of course, but are spread all over the code. It took us over a year to fix this kind of bugs but after that we could finally concentrate on our primary goal: the implementation of different optimisation methods. Today, you can choose from 11 different methods.

2 The methods

We have selected a set of 10 most promising optimisation methods from theory and we have topped the selection with a cost function evaluation option. The later could be considered a brute force optimisation algorithm and is intended for plotting up to three dimensional parameter spaces in *Mathematica*. With more dimensional cost functions the computation effort becomes intolerable however.

- Steepest descent algorithm
- Newton's method
- Davidon-Fletcher-Powell's method [1, 2]
- Monte Carlo algorithm
- Grid search algorithm
- Coordinate axes search algorithm
- Powell's method [3]
- Hooke-Jeeves' method [4]
- Constrained simplex method [5]
- A simple genetic algorithm [6]
- Cost function evaluation over the entire parameter space

We were very careful not to change any *SPICE* or *Nutmeg* command in our compilation, since they have

* Fakulteta za elektrotehniko, Univerza v Ljubljani, Ljubljana, Slovenija

become an unofficial standard in circuit analysis. However we had to add at least one command - *optimize*. Its syntax is consequently rather complex:

```
optimize [command]
command =
  analysis [number delete |
            number expression]
  | cost [expression]
  | implicit [number delete |
              number expression]
  | method [definition]
  | parameter [number delete |
      number [element name] [parameter name]
      [low value] [high value]
      [initial value]]
  | number_of_iterations value
definition =
  steepest_descent [r value{1.5}]
      [method quadratic | golden | fibonacci]
      [epsilon value{0.1}]
      [transformation no | sin | atcctg]
      [number_of_iterations value{100}]
      [gradient0 expression]
      [gradient1 expression]
      [...]
  | newton [r value{1.5}]
      [method quadratic | golden | fibonacci]
      [epsilon value{0.1}]
      [number_of_iterations value{100}]
  | davidon_fletcher_powell [r value{1.5}]
      [method quadratic | golden | fibonacci]
      [epsilon value{0.1}]
      [number_of_iterations value{100}]
      [modification no | modified | first |
                    second]
      [gradient0 expression]
      [gradient1 expression]
      [...]
  | monte_carlo [r value{1.5}]
      [method no | quadratic | golden |
              fibonacci]
      [epsilon value{0.1}]
      [number_of_iterations value{100}]
  | grid_search [level 2 | 3]
      [alpha value{1.3}]
      [epsilon value{0.1}]
  | axis_search [r value{1.5}]
      [method quadratic | golden | fibonacci]
      [epsilon value{0.1}]
      [number_of_iterations value{100}]
  | powell [r value{1.5}]
      [method quadratic | golden | fibonacci]
      [epsilon value{0.1}]
      [number_of_iterations value{100}]
  | hooke_jeeves [alpha value{1.3}]
      [epsilon value{0.1}]
  | complex [k value]
      [alpha value{1.3}]
      [size value{0.1}]
  | genetic [popsize value{10}]
      [lchrom value{1}]
      [maxgen value{19}]
      [pcross value{0.6}]
      [pmutation value{0.03}]
      [scalling value{2}]
  | parameter_space
      [outfile filename{opt.out}]
      [npts0 value{2}]
      [npts1 value{2}]
      [...]
```

Instead of going into more details we will now present some results of our optimisation tool.

3 Five Cases

One major difficulty with general analogue optimisation is the lack of good benchmark cases with known results. If the problem is simple enough, the cost function can be evaluated (maybe even plotted) in a very dense grid so the behaviour of different optimisation methods can be tested.

In this way we have verified our optimisation tool, but the really interesting question is, will it work with extremely large problems? Suppose we have a circuit with a cost function of 20 unknown variables. In order to survey at least some properties of the cost function we would have to analyse the circuits in some 10 points per variable, necessitating 10^{20} *SPICE* runs! In other words, there is no general way to know how an arbitrary cost function looks like, much less to discover its global minimum. So we made the following assumption. In theory many circuits are considered optimal for certain purposes. If these purposes are correctly translated into a proper cost function, then the optimisation process should converge to the known circuit parameters.

Here is our test circuit selection. For circuit specifics please refer to our homepage..

1. Schmidt trigger. This is a simple circuit with two bipolar transistors and a 2 dimensional cost function defining the edge points of the hysteresis. There is one simple implicit constrains, the cost function can be plotted.
2. Transistor amplifier. Another simple problem involving a transistor amplifier stage with a 2 dimensional cost function, which is looking for the most linear response with an implicitly constrained minimal gain factor.
3. Triangular to sine wave converter. The converter consists of a diode cascade approximating the sine wave with 7 linear segments. The optimal segment distribution is known from theory. The cost function is 12 dimensional.
4. Fifth order analogue filer. Again the optimal result is known from theory and should yield an elliptic pole distribution for the given circuit topology. This is a 15 dimensional optimisation case.
5. Bass reflex loudspeaker box. The problem of an optimal bass reflex enclosure and radiation vent is considered. The cost function is only 3 dimensional but the parameter space is heavily implicitly restricted.

Table 1: Summary of results

	Number of iterations					Minimal cost function value				
Circuit number	1	2	3	4	5	1	2	3	4	5
From theory								32µ	0.084	
Steepest descent	29	60	114	137	4	0.0175	0.3260	74µ	63.800	12.2100
Newton's method	7	7	411	137	10	0.2270	0.9010	498µ	67.400	12.2100
Davidon-Fletcher-Powell	30	51	301	89	4	0.0175	0.3360	38µ	64.000	12.2100
Monte Carlo	150	200	4000	8000	1000	0.0064	0.0009	38µ	8.910	0.7477
Grid search	121	173	4092	7764	265	0.0159	0.0008	11µ	0.920	0.9632
Axes search	50	51	1430	6280	89	0.0064	0.0009	21µ	18.800	0.9645
Powell's method	54	57	713	1590	92	0.0064	0.0009	21µ	23.300	0.9645
Hooke-Jeeves' method	69	103	1320	1498	233	0.0008	0.0011	9µ	1.940	0.9630
Complex	69	68	3931	2195	168	0.0031	0.0008	6µ	0.000	0.5643
Genetic	168	224	4224	8190	110	0.0056	0.1140	35µ	2.540	0.9007
Parameter space	10201	10201			8000	0.0005	0.0008			1.1045

We have run all eleven optimisation methods on all five test cases. The optimisations were done in all cases with default parameters and with the same initial parameter values. There was absolutely no fine-tuning of the different methods in order to keep the comparison fair. Nevertheless, it is very difficult to judge the methods by their default parameters and by a common initial parameter vector.

Table 1 shows a rough summary of our results. Not all methods have reached the same local minimum, as can be seen from the cost function columns. The most robust methods for our five cases are the Constrained simplex and Hooke-Jeeves' method. To show the complexity of the test cases we shall examine just the one method/circuit combination, which is marked in table 1.

4 The fifth order filter

Let us consider in detail the optimisation of the fifth order analogue filter with the constrained simplex algorithm. The circuit topology is presented in Figure 1. We set all capacitances to the same value 2.7nF. The values of all 15 resistors are subject to optimisation as can be seen from table 2.

The cost function is defined by the boundaries in Figure 1. We tolerate a 0.7dB ripple, further we demand at least 60dB damping and a cut-off bandwidth from 750Hz and 1KHz. The cost function value is actually the integral of the *ac* response outside these limits. The table 2 summarises the explicit constraints and the initial values of each parameter. After 2195 iterations the complex method stops with a zero valued cost function and optimal resistor values which are quite different from the expected theoretically determined values.

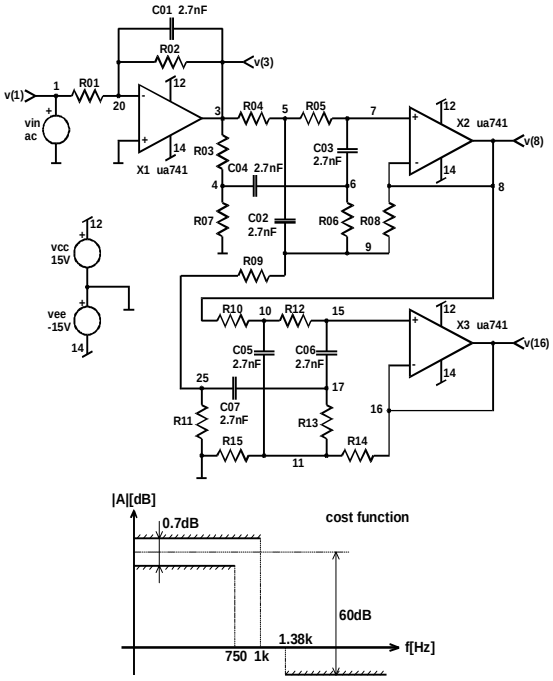

Figure 1: Active low-pass RC filter

Let us examine the Bode response of the initial iteration (marked as unoptimised), the optimised circuit and the theoretically designed parameters (marked as analytic) in Figure 2.

Although the optimal resistor values are quite different from the theoretically determined the cost function is even a bit lower than expected. The *ac* response of the optimised filter accomplishes all requests determined in cost function. Theoretical and optimised filter have equivalent *ac* responses. The only difference is slightly higher damping factor at the optimised filter.

Table 2: Resistor values (values are rounded)

Par.	Exp. Constr.	Initial	Optimal	Theoret.
R01	10k ÷ 100k	50k	29.7k	19.6k
R02	100k ÷ 1meg	500k	183k	196k
R03	500 ÷ 5k	2.5k	2.21k	1k
R04	50k ÷ 500k	250k	293k	147k
R05	100k ÷ 1meg	500k	187k	154k
R06	10k ÷ 100k	50k	16.2k	37.4k
R07	10 ÷ 100	50	36.1	71.5
R08	100 ÷ 1k	500	212	260
R09	100 ÷ 1k	500	732	740
R10	50k ÷ 500k	250k	200	110k
R11	100 ÷ 1k	500	390	402
R12	50k ÷ 500k	250k	55.7k	110k
R13	10k ÷ 100k	50k	24.7k	27.4k
R14	10 ÷ 100	50	38.6	40
R15	500 ÷ 5k	2.5k	2.3k	960

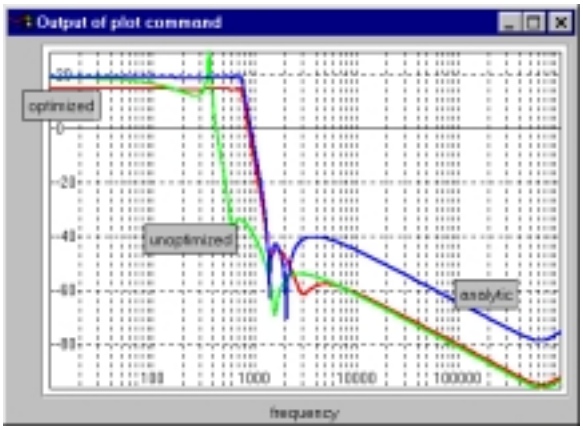

Figure 2: The Bode response of the filter with initial, optimal and theoretical values of resistors.

5 Conclusions

The implemented optimisation methods have been tested on five different circuits. The outcome was not always successful which is due to the default parameters of each method. We have also spent a lot of time tuning the parameters of all methods and searching for more convenient initial points. We have also combined several methods; for example we have used the Genetic algorithm to find a suitable initial point for the Steepest descent algorithm. It is our conclusion, that almost any optimisation method can be successfully used on any problem, providing the user has some feeling for the nature of the circuit as well as for the properties of the employed optimisation method.

The later assumptions is actually rather demanding. Our optimisation command is far from being foolproof. In order to optimise complex circuits, the user must first study the particular optimisation method from theory. Only then he will fully understand all the *optimize* command options and be able to use them precisely.

Each of the optimisation cases in table 2 has been completed in less 10 minutes. However, with large circuits, complex cost functions and a large number of optimisation parameters the entire optimisation process may become very time and memory consuming.

[1] W. C. Davidon, "*Variable metric method of minimisation*", Report ANL-5990 (rev.), Argonne National Laboratory, Argonne, Ill., 1959

[2] R. Fletcher, M. J. D. Powell, "*A rapidly convergent descent method for minimisation*", Computer Journal, Volume 6, pages 163 - 168, 1963

[3] M. J. D. Powell, "*An efficient method for finding the minimum of a function of several variables without calculating derivatives*", Computer Journal, Volume 7, pages 155 - 162, 1964

[4] R. Hooke, T. A. Jeeves, "*"Direct search" solution of numerical and statistical problems*", Journal of ACM, Volume 8, pages 212 - 229, 1961

[5] M. J. Box, "*A new method of constrained optimisation and comparison with other methods*", Computer Journal, Volume 7, pages 42 – 52, 1965

[6] David E. Goldberg, "*Genetic Algorithms in Search, Optimisation & Machine Learning*", Addison-Wesley Publishing, 1989